```
// Computer Program Listing Appendix Under 37 CFR 1.52(e)


/*
  Appendix includes two sets of code samples, with the first set including:
* httprequest_dorequest
  HTTPRequest::do_request method.
* httprequest_doqueryrequest
  HTTPRequest::DoQueryRequest method.
  HTTPRequest::do_request does the work of connecting
  to the appropriate database, and executing the appropriate
  service.
* httprequest_parseuri
  HTTPRequest::ParseURI method.
* httprequest_serviceexists
  HTTPRequest::ServiceExists method.
  HTTPRequest::ParseURI and HTTPRequest::ServiceExists
  do most of the work of parsing a URI to determine which
  service maps to that particular URI.
* httpconnection_decl
  Class declaration for HTTPConnection.
* httpprotocol_decl
  Class declaration for HTTPProtocol.
* httprequest_decl
  Class declaration for HTTPRequest.
* httppres_decl
  Class declaration for HTTPPres.
*/

// httpconnection_decl
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
class HttpConnection {
  public:
    HttpConnection *  _next;
  protected:
    HttpSocket *  _socket;
    HttpSockOStream *  _stream;
    HttpConnectionState  _state;
    HttpProtocol *  _protocol;
    HttpRequest *  _request;
    HttpListener *  _listener;
    HttpString    _rmt_addr;
    HttpString    _lcl_addr;
    HttpString    _rxline;
    char *    _dbname;
    HttpString    _dbConnected;
    char *    _rxbuffer;
    char *    _decrypt_buffer;
    char    _last;
    HttpRxCompletion *  _rxcomplete;
    HttpTxCompletion *  _txcomplete;
    uint32    _request_size;
    uint32    _pkts_received;
    a_fast_tod    _last_read_time;
    a_bool    _ignore_receive;
    a_bool    _dbname_required;
    a_bool    _dbname_provided;
    a_web_protocol_type  _type;
  public:
    HttpConnection( SysSocket sock, char *dbn, a_web_protocol_type type,
        HttpListener *l, char *lcl_addr, char *rmt_addr );
    ~HttpConnection();
    void    Start( void );
    void    Stop( a_bool on_worker = FALSE );
    void    DelayedStop( void );
    void    RequestFinished( void );
    void    CleanUp( void );
    void    ProcessSend( int err, int datalen );
    a_bool    ProcessLine( char * rxbuffer, int * offset, int datalen );
    void    ProcessRecv( int err, int datalen );
    void    ProcessData( char *, int );
    void    ProcessHttpsRecv( a_bool force = FALSE );
    void    IgnoreReceive();
    a_web_protocol_type  GetType( void ) const { return _type; }
    char *    GetDbName( void ) const { return _dbname; }
    a_bool    DBNameRequired( void ) const { return _dbname_required; }
    a_bool    DBNameProvided( void ) const { return _dbname_provided; }
    char *    GetDbConnected( void ) const { return _dbConnected.c_str(); }
    void    SetDbConnected( char *str ) {
  _dbConnected.clear();
  _dbConnected.append( str );
    }
    HttpListener *  GetListener( void ) const { return _listener; }
    HttpProtocol *  GetProtocol( void ) const { return _protocol; }
    HttpRequest *  GetRequest( void ) const { return _request; }
    HttpOrderedList *  GetVariables( void ) const { return _request->GetVariables(); }
    a_bool    ParseRequestString( HttpRequest *request, HttpString *str );
    a_bool    ParseHeaderString( HttpRequest *request, HttpString *str );
    a_bool    ParseURI( HttpRequest * request );
    a_bool    ParseVersion( HttpRequest * request );
    a_bool    CanDelete( void );
    a_bool    CheckForTimeout( void );
  a_bool   SendHttpHeaders( HttpRequest *request, HttpOStream *stream );
  a_bool   SendHttpError( HttpRequest *request, HttpOStream *stream );
  a_bool   SendSQLError( HttpRequest *request, HttpOStream *stream );
  a_bool   IsSecure( void ) const;
  void   UpdateReceivedConnProperties( p_Connection ) const;
  void   UpdateSentConnProperties( p_Connection ) const;
  void   GetRemoteMachineAddr( char *buf, int32 buflen );
   uint32   GetIdleTimeout( void ) const;
   a_fast_tod   GetLastRequestTime( void ) const { return _last_read_time; }
   HttpString   *GetLocalMachineAddr( void )  { return &_lcl_addr; }
   int64   GetBytesWritten( void ) const  { return _socket->getBytesWritten(); }
 protected:
   HttpSockOStream *  GetStream() const     { return _stream; }
   friend class HttpListener;
   friend class HttpProtocol;
private:
   void   StartRequest();
};


// httppres_decl
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
class HttpPres : public StubPres {
 protected:
   HttpOStream *      _ostream;
   UTCollation *  _col;      // db's collation
   HttpPresStatus  _status;
   HttpOrderedList *  _args;
   uint32   _arg_id;
   uint32   _arg_len;
   a_byte *    _arg_data;
   uint32   _descriptor_count;  // number of columns in query
   uint32   _row_ctr;   // number of rows written
   uint32   _col_ctr;   // number of columns described/written
   a_ptrint    _bin_bits;    // which columns are binary data
   a_bool    _printed_doc;    // has any doc been written
   a_bool    _started_doc;    // has the doc been started
   a_bool    _started_set;    // has started the result set
   a_bool    _started_row;
   a_bool    _wants_null_values;  // output wants to show nulls
   a_bool PutData( char ** buf, size_t len, uint32 flags );
   a_bool PutData( char * buf, size_t len, uint32 flags )
 { return PutData( &buf, len, flags ); }
   // Put data Encoded & Charset-converted
   inline a_bool PutEC( char * buf )
```

```
{ return PutData( buf, _strlen(buf), HF_ENC|HF_CONV ); }
  inline a_bool PutEC( char ** buf, size_t len )
{ return PutData( buf, len, HF_ENC|HF_CONV ); }
  inline a_bool PutEC( char * buf, size_t len )
{ return PutData( buf, len, HF_ENC|HF_CONV ); }
  inline a_bool PutEC( HttpString & str )
{ return PutData( str.str(), str.length(), HF_ENC|HF_CONV ); }
  // Put data Charset-converted (no encoding)
  inline a_bool PutCC( char ch )
{ return PutData( &ch, 1, HF_CONV ); }
  inline a_bool PutCC( char * buf )
{ return PutData( buf, _strlen(buf), HF_CONV ); }
  inline a_bool PutCC( char * buf, size_t len )
{ return PutData( buf, len, HF_CONV ); }
  inline a_bool PutCC( HttpString & str )
{ return PutData( str.str(), str.length(), HF_CONV ); }
  // Put ASCII data - not encoded, and not Charset-converted unless it needs
  // to be (i.e. output character set is multi-byte)
  inline a_bool PutAsc( char c )
{ return PutData( &c, 1, HF_ASC ); }
  inline a_bool PutAsc( char * buf )
{ return PutData( buf, _strlen(buf), HF_ASC ); }
  inline a_bool PutAsc( char * buf, size_t len )
{ return PutData( buf, len, HF_ASC ); }
  inline a_bool PutAsc( HttpString & str )
{ return PutData( str.str(), str.length(), HF_ASC ); }
public:
  HttpPres( HttpOStream * ostream, UTCollation * col )
: StubPres()
, _ostream( ostream )
, _col( col )
, _status( PRES_OK )
, _args( NULL )
, _arg_id( 0 )
, _arg_len( 0 )
, _arg_data( NULL )
, _descriptor_count( 0 )
, _row_ctr( 0 )
, _col_ctr( 0 )
, _bin_bits( 0 )
, _printed_doc( FALSE )
, _started_doc( FALSE )
, _started_set( FALSE )
, _started_row( FALSE )
, _wants_null_values( TRUE )
  {
  }
  virtual ~HttpPres();
  inline HttpPresStatus GetPresStatus( void )
  {
return _status;
  }
  inline a_bool PresStatusOk( void )
  {
return _status == PRES_OK;
  }
  void SetArguments( HttpOrderedList * args )
  {
// Note: caller responsible for freeing args.
_args = args;
  }
  a_bool  ReceiveHostVariable( an_sqlpres_value *value, uint32 *index );
  a_bool  ReceiveMultiBegin( uint32* total_length );
  a_bool  ReceiveMultiPiece( void * buff, uint32 buff_len, uint32* recv_len );
  a_bool  ReceiveMultiEnd( void );
  void  ReportSQLError( HttpProtocol * proto );
protected:
  a_bool IsBinaryColumn( uint32 c );
  inline void SetPresStatus( HttpPresStatus status )
  {
if( _status == PRES_OK ) {
    _status = status;
}
  }
  void      MakeErrorString( char * buff, size_t len, a_bool replace_quotes );
public:
  // methods inherited from StubPres
  a_bool  SendValueSetDescriptor( uint16    desc_id,
        char     *coln_name,
        uint16    coln_namelen,
        char     *table_name,
        char     *db_name,
        char     *user_name,
        a_byte    asa_domain_id,
        uint32    asa_usertype,
        uint32    asa_flags,
        uint32    asa_maxlen,
        uint16    asa_prec,
        uint16    asa_scale,
        a_describe_flag  describe_flags );
  a_bool  ReceiveDescriptor( an_sqlpres_desc * desc );
  a_bool  SendValue( a_domain_number domain_id, void * data, uint32 len, uint32 truelen, a_textptr_value * textptr );
  a_bool  SendNullValue( a_domain_number domain_id, p_expr expr );
  a_bool  SendNoneValue( a_domain_number domain_id );
  a_bool  SendMultiBegin( a_domain_number domain_id, uint32
total_length, uint32 untruncated_length, a_textptr_value * textptr, uint32 flags );
    a_bool  SendMultiPiece( void ** data, uint32 piece_length, uint32 flags );
    a_bool  SendMultiEnd( uint32 flags );
    a_bool  SendValueSetRow( an_error_mapping *errmap, an_sqlpres_tran_status tran_status );
    a_bool  SendSuccessOrError( p_Connection  conn,
            a_bool    send_iocount,
            a_bool    send_tran_flags );
    a_bool  SendRequestDone( void );
    a_bool  SyncPoint( void );
    class DBConnConverter *GetConverter( void ) {
  return _ostream->getConverter();
    }
    const class CharsetInfo *GetCharsetInfo( void ) {
  return _ostream->getCharsetInfo();
    }
  protected:
    // methods that control output
    virtual void AddColumn(
      char *    table_name,
      char *    coln_name,
      uint16    coln_namelen,
      uint32    asa_usertype )
    {
  _unused( table_name );
  _unused( coln_name );
  _unused( coln_namelen );
  _unused( asa_usertype );
    };
    virtual void BeginDoc( void ) {};    // output doc header
    virtual void BeginResultSet( void ) {};  // start of result set
    virtual void BeginRow( void ) {};
    virtual void BeginColumn( void ) {};
    virtual void EndColumn( void ) {};
    virtual void EndRow( void ) {};
    virtual void EndResultSet( void ) {};
    virtual void EndDoc( void ) {};
    virtual void SendColumnValue( void * data, uint32 len ) { _unused(data); _unused(len); };
    virtual void SendColumnNull( void ) {};
    virtual void SendColumnNone( void ) {};
    virtual void SendColumnMultiBegin( void ) {};
    virtual void SendColumnMultiPiece( void ** data, uint32 len ) { _unused(data); _unused(len); };
    virtual void SendColumnMultiEnd( void ) {};
    virtual void SendSQLError( void * errmsg, size_t len ) = 0;
    virtual void NoContentDocBody( void ) {};  // called when doc has no content
```

```
private:
  // these routines guarantee that the virtual versions are called in correct order
  inline void DoBeginDoc( a_bool starting_a_row )
  {
if( ! _started_doc ) {
    BeginDoc();
    if( starting_a_row ) {
  BeginResultSet();
  _started_set = TRUE;
    }
    _started_doc = TRUE;
    _printed_doc = TRUE;
}
  }
  inline void DoBeginRow( void )
  {
if( ! _started_row ) {
    DoBeginDoc( TRUE );    // ensure we've started the doc
    BeginRow();
    _started_row = TRUE;
    _col_ctr     = 0;
    _row_ctr ++;
}
  }
  inline void DoBeginColumn( void )
  {
DoBeginRow();      // ensure we have started a row
BeginColumn();
  }
  inline void DoEndColumn( void )
  {
EndColumn();
_col_ctr ++;
  }
  inline void DoEndRow( void )
  {
if( _started_row ) {
    EndRow();
    _started_row = FALSE;
}
  }
  inline void DoEndResultSet( void )
  {
if( _started_set ) {
    DoEndRow();      // ensure we have closed the row
    EndResultSet();
    _started_set = FALSE;
}
  }
```

```
    inline void DoEndDoc( void )
    {
  if( _printed_doc ) {
      // we have something on the document
      DoEndResultSet();
  } else {
      // we never generated any doc content
      if( ! _started_doc ) {
    DoBeginDoc( FALSE );
      }
      NoContentDocBody();
  }
  if( _started_doc ) {
      EndDoc();
      _started_doc = FALSE;
  }
    }
};


// httpprotocol_decl
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
class HttpProtocol {
  private:
    HttpPhase    _phase;    // current phase of the request
    a_perf_tod    _tod_connected;  // time of day client connected
    a_perf_tod    _tod_queued;  // time of day request was queued
    a_perf_tod    _tod_started;  // time of day request was started
//  a_perf_tod    _tod_finished;  // time of day request was finished
= time it is logged
    HttpString    _method;
    HttpString    _uri;
    HttpString    _version;
    HttpHashTable  _request;  // table of headers in request
    HttpHashTable  _response;  // table of headers to send in response
    HttpHashTable  _options;  // table of options
    HttpString    _lastHeaderKey;  // if value is continued on next
line,
                                  // we need to know what to
append it to
    HttpStatus    _status;
    HttpString    _errorstr;
    /*
  _body_expected is whether or not we're expecting a body with this
request.
  _body_expected_length is the expected length of the body (based on
HTTP headers).
  _body.length() is the actual length of the body received so far.
    */
    HttpString    _body;
  a_bool    _body_expected;
  uint32    _body_expected_length;
  HttpConnection *  _connection;
  HttpLogger *  _logger;
  a_bool    _has_been_logged;
  void    SetDateHeaders( void );
  a_bool    _send_headers;
  a_bool    _send_body;
  a_bool    _content_type_set;
  friend class HttpRequest;
  friend class HttpLogger;
public:
  HttpProtocol( HttpConnection * connection, HttpLogger * logger );
  ~HttpProtocol();
  HttpPhase    GetPhase( void )    { return( _phase ); }
  a_perf_tod *  GetTodConnected( void )    { return( &_tod_connected ); }
  a_perf_tod *  GetTodQueued( void )    { return( &_tod_queued ); }
  a_perf_tod *  GetTodStarted( void )    { return( &_tod_started ); }
  void    SetReqQueued( void );
  void    SetReqStarted( void );
  void    SetReqFinished( void );
  void    SetOkToDelete( void );
  const HttpString *  GetMethod() const    { return &_method; }
  const HttpString *  GetUri() const      { return &_uri; }
  const HttpString *  GetVersion() const    { return &_version; }
  /*
Request header methods
  */
  void    SetRequestHeader( const char    *key,
      const HttpString  *value ) {
_request.Set( key, value->str(), value->length() );
_lastHeaderKey.clear();
_lastHeaderKey.append( key );
  }
  void    SetRequestHeader( const char  *key,
      const char  *value,
      const size_t  len ) {
_request.Set( key, value, len );
_lastHeaderKey.clear();
_lastHeaderKey.append( key );
  }
  a_bool    AppendRequestHeader( HttpString  &value );
  HttpString *  GetRequestHeader( const char * key ) {
return _request.Get( key );
  }
  HttpString *  GetRequestNextKey( const char * key ) {
return _request.GetNextKey( key );
  }
```

```
    /*
  Response header methods
    */
    void    SetResponseHeader( const char * key, const HttpString * value )
                { _response.Set( key, value->str(), value->length() ); }
    a_bool    SetResponseHeader( const char * key, const char * value );
    HttpString *  GetResponseHeader( const char * key )  { return _response.Get( key ); }
    a_bool    SetHTTPOption( char * optname, char * value );
    HttpString *  GetHTTPOption( char * optname )    { return( _options.Get( optname ) ); }
    a_bool    ContentTypeSet( void ) const { return _content_type_set; }
    // _status methods
    void    SetHttpStatus( HttpStatus status )  { _status = status; }
    void    SetHttpStatus( HttpRequestState state );
    HttpStatus    GetHttpStatus( void ) const    { return _status; }
    char *    GetHttpStatus( char * buf, size_t len );  // get status string
    void    SetErrorString( const char * str, const size_t len );
    const HttpString *  GetErrorString( void ) const    { return &_errorstr; }
    // _body methods
    HttpString *  GetBody( void )       { return &_body; }
    a_bool    GetBodyExpected( void ) const    { return _body_expected; }
    uint32    GetBodyExpectedLength( void ) const  { return _body_expected_length; }
    a_bool    ParseRequest( const HttpString * request );
    a_bool    ParseHeader( const HttpString * header );
    a_bool    ParseMethod( void );
    a_bool    ParseBodyLength( void );
    a_bool    SendHttpHeaders( HttpOStream * stream );
    a_bool    SendHttpError( HttpOStream * stream );
    void    WriteLogEntry( void );
    void    CleanUp( void );
    a_bool    ShouldSendBody( void ) const { return _send_body; }
#if !PRODUCTION
  private:
    HttpString    _resbody;
  public:
    HttpString *  GetResBody( void ) { return &_resbody; }
#endif
};


// httprequest_decl
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
class HttpRequest : public RQBaseItem
{
    HttpConnection *  _connection;
    HttpProtocol *  _protocol;
    HttpOStream *  _stream;
    HttpService *  _service;
    HttpService *  _dservice;
    Database *    _db;
    p_Connection  _dbconnection;
    p_Worker    _worker;
    a_bool    _cancel;
    uint32    _uid;
    HttpString *       _parms;
    HttpRequestState  _state;
    HttpString    _username;
    HttpString    _password;
    HttpString    _database;
    HttpString    _service_name;
    HttpString    _arguments;
    HttpString    _url_path;
    HttpOrderedList  _variables;
    a_bool    _headers_sent;
public:
    HttpRequest( HttpConnection * c, HttpProtocol * p, HttpOStream * s );
    ~HttpRequest();
    p_Worker    GetWorker() { return _worker; }
    HttpOrderedList *  GetVariables() { return &_variables; }
    a_bool    Connected() { return _dbconnection != NULL; }
    virtual void  do_request();
    void    Cancel();
    void    CleanUp();
private:
    a_bool      ServiceExists( HttpString & name );
    a_bool      DetermineServiceOptions();
    a_bool      ProcessAuthentication();
    a_bool      ProcessHttpAuthentication();
    a_bool      ProcessBasicAuthentication( const HttpString * base64_credentials );
    a_bool      DatabaseConnect( HttpString &charset );
    void      DatabaseDisconnect();
    a_bool      ParseURI();
    a_bool      ParseArgs( HttpHashTable * argtable, HttpString * args );
    a_bool      ParseMultipartFormData( HttpHashTable * argtable, HttpString * args, char * boundary );
    a_bool      ParseArguments( HttpHashTable * arg_table );
    a_bool      ParseBodyArguments( HttpHashTable * arg_table );
    a_bool      GetURLPathArguments( HttpHashTable * argtable );
    a_bool      DoDishRequest();
    a_bool      DoQueryRequest();
    a_bool      DoWSDLRequest();
    a_bool      ParseSoapRequest();
    void      RedirectToSecure();
    void      UpdateReceivedConnProperties( void );
    void      UpdateSentConnProperties( void );
    void      ReportSQLError( HttpPres * pres );
    void      SendHttpHeaders();
    void      MakeURI( HttpString * host, HttpService * svc, HttpString & uri );
};


// httprequest_doqueryrequest
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
a_bool HttpRequest::DoQueryRequest()
/*********************************/
{
    p_Connection    dbc = _CurrentConnection;
    p_statement      stmt = NULL;
    HttpPres *      pres = NULL;
    p_cursor      crsr = NULL;
    p_cursor      real_crsr = NULL;
    p_stmt      s = NULL;
    p_expr      expr = NULL;
    a_bool      variable;
    HttpOrderedList stmt_parms;
#define CHECK_CANCEL()   if( _cancel ) { goto cleanup; }
    if( _service->GetServiceType() == HTTP_SERVICE_SOAP ) {
   if( !ParseSoapRequest() ) {
       _state = REQUEST_BAD_REQUEST;
       return FALSE;
   }
    } else {
   if( !GetURLPathArguments( &_variables ) ) {
       _state = REQUEST_BAD_REQUEST;
       return FALSE;
   }
   if( !ParseArguments( &_variables ) ) {
       _state = REQUEST_BAD_REQUEST;
       return FALSE;
   }
   if( !ParseBodyArguments( &_variables ) ) {
       _state = REQUEST_BAD_REQUEST;
       return FALSE;
   }
    }
    CHECK_CANCEL();
    /*
```

```
Set up the presentation layer
 */
 switch( _service->GetServiceType() ) {
case HTTP_SERVICE_XML:
   pres = New_HttpPresXML( _stream, dbc->db()->collation );
   break;
case HTTP_SERVICE_HTML:
   pres = New_HttpPresHTML( _stream,
        dbc->db()->collation,
        ( _parms == NULL
          ? _arguments : *_protocol->GetUri() )
       );
   break;
case HTTP_SERVICE_RAW:
   pres = New_HttpPresRaw( _stream, dbc->db()->collation );
   break;
case HTTP_SERVICE_SOAP:
   {
 HttpString  nspace;
 HttpString  opname;
 if( _dservice != NULL ) {
     MakeURI( (HttpString *)_protocol->_request.Get( "Host" ), _dservice, nspace );
 } else {
     MakeURI( (HttpString *)_protocol->_request.Get( "Host" ), _service, nspace );
 }
 GetOpName( _dservice, _service, opname  );
 pres = New_HttpPresSOAP( _stream, dbc->db()->collation, &nspace, opname.c_str() );
   }
   break;
default:
   _assertD( FALSE );
   break;
 }
 dbc->pres = pres;
 /*
Prepare the statement
 */
 if( _parms == NULL ) {
// arbitrary query is allowed
expr = an_ExprBuilder::GblBuilder.DB_Expr_str_len( _arguments.c_str(), (a_row_length) _arguments.length() );
 } else {
// service specifies query
expr = an_ExprBuilder::GblBuilder.DB_Expr_str_len( _parms->c_str(), (a_row_length)_parms->length() );
 }
 if( expr == NULL ) {
```

```
_state = REQUEST_INTERNAL_ERROR;
goto cleanup;
  }
  stmt = PrepareExpr( expr, GOAL_STATEMENT, NULL, FALSE );
  if( stmt == NULL ) {
DE_Free_expr( expr );
_assertD( SQLErr( dbc ) );
ReportSQLError( pres );
goto cleanup;
  }
  dbc->SetLastStatement( expr );
  DE_Free_expr( expr );
  if( stmt->type != STMT_SELECT && stmt->type != STMT_CALL ) {
a_heap_ref  ref;
ref.mem = stmt;
DV_Free_heap( &ref );
_state = REQUEST_BAD_REQUEST;
goto cleanup;
  }
  if( !SetArgumentNames( &stmt_parms, stmt ) ) {
a_heap_ref  ref;
ref.mem = stmt;
DV_Free_heap( &ref );
_state = REQUEST_SQL_ERROR;
goto cleanup;
  }
  stmt_parms.CopyValues( &_variables );
  CHECK_CANCEL();  // last change to check cancel before doing actual work
  /*
Set up a cursor
  */
  CreatePreparedStatement( dbc, stmt, &s );
  _assertD( !SQLErr( dbc ) );
  _assertD( s != NULL );
  dbi_describe_statement( s, DESCT_SELECTLIST, DESC_NO_FLAGS, 0, &variable );
  pres->SetArguments( &stmt_parms );
  crsr = dbc->add_cursor( "http_cursor" );
  crsr->stmt = s;
  DBOpenCursor( crsr, -1, CURSOR_READONLY, TRUE );
  if( crsr->ref != NULL && !SQLErr( dbc ) ) {
// execute a procedure
dbi_resume_procedure( crsr->ref, TRUE );
if( SQLErr( dbc ) ) {
    // proc_cursor has been closed and freed due to error
    crsr->proc_cursor = NULL;
    crsr->stmt = NULL;
}
  }
```

```
    real_crsr = dbc->FindRealCursor( crsr, NULL, FALSE );
    _state = REQUEST_SUCCESS;
    if( real_crsr == NULL ) {
  // If statement was procedure call/batch, it might not have had
  // a result set.
  if( SQLErr( dbc ) ) {
      ReportSQLError( pres );
  } else {
      SendHttpHeaders();
  }
    } else {
  dbc->lock_cursor( real_crsr, TRUE );
  dbi_row_descriptor( (p_db_cursor)real_crsr->db_cursor.mem );
  /*
      Update the status of the HTTP connection and send the headers
      out over the wire.
  */
  _assertD( _state == REQUEST_SUCCESS );
  SendHttpHeaders();
  if( _protocol->ShouldSendBody() ) {
      DoFullFetch( real_crsr );
      if( SQLErr( dbc ) ) {
    ReportSQLError( pres );
      }
      switch( pres->GetPresStatus() ) {
      case PRES_OSTREAM_ERROR:
    _state = REQUEST_FAILURE;
    break;
      case PRES_NO_XML_USERTYPE:
      case PRES_NOT_XML_RESULTSET:
    _state = REQUEST_NOT_XML;
    break;
      }
  }
  dbc->unlock_cursor( real_crsr );
    }
    DoCloseCursor( crsr );
    dbc->drop_statement( s );
    if( _protocol->ShouldSendBody() ) {
  dbc->pres->SendSuccessOrError( dbc, TRUE, TRUE );
    }
#undef CHECK_CANCEL
 cleanup:
    if( pres != NULL ) {
  dbc->pres = NULL;
  delete pres;
    }
    if( _cancel ) {
  _state = REQUEST_CANCELED;
    }
    return _state == REQUEST_SUCCESS;
}


// httprequest_dorequest
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
void HttpRequest::do_request()
/****************************/
{
    UserDef  *user = NULL;
    HttpString  charset( 100 );
    _worker = _CurrentWorker;
    _protocol->SetReqStarted();
#define CHECK_CANCEL()   if( _cancel ) { goto finish; }
    CHECK_CANCEL();
    if( !ParseURI() ) {
  _state = REQUEST_BAD_REQUEST;
  goto finish;
    }
    if( !DatabaseConnect( charset ) ) {
  goto finish;
    }
    if( !DetermineServiceOptions() ) {
  goto finish;
    }
    /*
  Depending on the URI, the DISH service either acts like a WSDL
  service or a SOAP service.
    */
    if( _service->GetServiceType() == HTTP_SERVICE_DISH ) {
  if( _arguments.eq( "wsdl" ) ) {
      /*
    "Fall through".  The DISH service is generating WSDL, and will be
    treated as a WSDL service below.
      */
  } else {
      /*
    Set _service_name to the service specified in SOAPAction.
    SOAPAction URLs have the format "http://hostname/dbname/path"
    (quotes included).
      */
      HttpString * action = _protocol->GetRequestHeader( "SOAPAction"
);
      if( action != NULL ) {
    HttpStrlStream  stream( action );
    int    slash = 0;
    unsigned char  c;
    while( TRUE ) {
        if( !stream.get( c ) ) goto finish;
        if( c == '/' ) {
```

```
    if( ++slash >= 4 ) break;
      }
  }
  _service_name.clear();
  while( TRUE ) {
      if( !stream.get( c ) ) goto finish;
      if( c == '"' ) break;
      _service_name.append( c );
  }
  _dservice = _service;
  if( !DetermineServiceOptions() ) {
      goto finish;
  }
  /*
      DISH won't act as a proxy for anything other than SOAP services.
  */
  if( _service->GetServiceType() != HTTP_SERVICE_SOAP ) {
      goto finish;
  }
    }
}
  }
  /*
Secure connection
  */
  if( _service->GetSecureRequired() && !_connection->IsSecure() ) {
RedirectToSecure();
goto finish;
  }
  /*
Authentication
  */
  if( _service->GetAuthRequired() ) {
if( !ProcessAuthentication() ) {
    goto finish;
}
  }
  user = FindUserByID( _uid );
  if( user == NULL ) {
_state = REQUEST_INVALID_USER;
goto finish;
  }
  _dbconnection->SetUser( user->GetSAUserName(), FALSE, TRUE );
  _dbconnection->set_user( user );
  if( AuditingOn( _db ) ) {
char      address[80];
a_web_protocol_type  proto = WEB_HTTP;
_connection->GetRemoteMachineAddr( address,
           (int32)sizeof( address ) );
```

```
if( _connection->IsSecure() ) {
    proto = WEB_HTTPS;
}
AuditHttpConnection( user->GetSAUserName(), address, GetProtoStr( proto ),
        TRUE, _db );
  }
  user->Release();
  CHECK_CANCEL();
  if( !CallLoginEnvironment( _dbconnection ) ) {
DB_Exec_connect_failed_event_handler( _db,
    (char *)_dbconnection->_ew_static_user.str(), NULL );
_state = REQUEST_INVALID_AUTHENTICATION;
goto finish;
  }
  if( Debug ) {
DB_Message( IDS_ENG_USER_CONNECTED_TO_DATABASE_FROM_HTTP,
      _dbconnection->handle(),
      _dbconnection->get_user()->name,
      _db->_ro_alias,
      GetProtoStr( _connection->GetType() ) );
DBConnConverter *conv = _stream->getConverter();
if( conv == NULL ) {
    DB_Message( IDS_ENG_CHARSET_TRANSLATION_ENABLED_NOT_NEEDED_WITH_CONNID,
          _dbconnection->handle(),
    _db->cs_info->sybase_label );
} else {
    const CharsetInfo *cli_cs_info =
  UTLocale::GetCharsetInfoFromAsaCID( conv->outbound().GetDestCid() );
    DB_Message( IDS_ENG_CHARSET_TRANSLATION_ENABLED_NEEDED_WITH_CONNID,
    _dbconnection->handle(),
    _db->cs_info->sybase_label,
    cli_cs_info->sybase_label );
}
  }
  DB_Exec_system_event_handler( _dbconnection->db(), _dbconnection, EVT_Connect );
  ProcDebug::ConnectionStarted( _dbconnection );
  /*
Define special request header fields so user can access them.
BUGBUG: What do we set URI to when processing a SOAP request through
  a DISH service?  For now, we're setting it to the URI of the
  DISH service, not the proxied SOAP service.
  */
  _protocol->SetRequestHeader( "@HttpMethod", _protocol->GetMethod() );
  _protocol->SetRequestHeader( "@HttpURI", _protocol->GetUri() );
```

```
    _protocol->SetRequestHeader( "@HttpVersion", _protocol->GetVersion() );
    UpdateReceivedConnProperties();
    CHECK_CANCEL();
    switch( _service->GetServiceType() ) {
  case HTTP_SERVICE_XML:
  case HTTP_SERVICE_HTML:
  case HTTP_SERVICE_RAW:
  case HTTP_SERVICE_SOAP:
      DoQueryRequest();
      break;
  case HTTP_SERVICE_DISH:
  case HTTP_SERVICE_WSDL:
      DoWSDLRequest();
      break;
  default:
      _assertD( FALSE );
      _state = REQUEST_INVALID_SERVICE;
      break;
    }
#undef CHECK_CANCEL
finish:
    DatabaseDisconnect();
    if( _cancel ) {
  _state = REQUEST_CANCELED;
    }
    if( _state != REQUEST_SUCCESS ) {
  _protocol->SetHttpStatus( _state );
  if( _state == REQUEST_SQL_ERROR ) {
  } else {
      _protocol->SendHttpError( _stream );
  }
    }
    _stream->flush();
    UpdateSentConnProperties();
    _worker = NULL;
    _connection->RequestFinished();
}


// httprequest_parseuri
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
a_bool HttpRequest::ParseURI()
/****************************/
{
    HttpStrlStream  stream( _protocol->GetUri() );
    unsigned char   c;
    a_bool      got_service = FALSE;
    _assertD( _database.length() == 0 );
    _assertD( _service_name.length() == 0 );
    _assertD( _arguments.length() == 0 );
    if( !stream.get( c ) || c != '/' ) {
  return FALSE;
    }
    if( _connection->DBNameProvided() ) {
  _database.append( _connection->GetDbName() );
    } else {
  while( TRUE ) {
      if( !stream.get( c ) ) return TRUE;
      if( c == '/' ) break;
      if( c == '?' ) {
    got_service = TRUE;
    break;
      }
      _database.append( c );
  }
    }
    if( !got_service ) {
  while( TRUE ) {
      if( !stream.get( c ) ) return TRUE;
      if( c == '?' ) break;
      _service_name.append( c );
  }
    }
    while( stream.get( c ) ) {
  _arguments.append( c );
    }
    return TRUE;
}


// httprequest_serviceexists
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
a_bool HttpRequest::ServiceExists( HttpString & name )
/****************************************************/
{
    a_bool        result = FALSE;
    HttpService *      svc;
    a_statement *      stmt;
    p_Database        db = _CurrentDB;
    svc = HttpService::Find( db, name.c_str() );
    if( svc == NULL ) {
  // The service by the full name does not exist...
  // need to split it up into <name>/<url> pieces
  char *        str   = name.c_str();
  size_t        len   = name.length();
  if( len == 0 ) {
      return FALSE;
  }
  size_t        split = len - 1;
for( ; split > 0; split-- ) {   // Note: first char cannot be '/'
    if( str[split] == '/' ) {
  // split at this point
  HttpString tname( str, split );
  svc = HttpService::Find( db, tname.c_str() );
  if( svc != NULL ) {
      if( svc->GetUrlPathType() != URL_PATH_OFF ) {
    // got [name=0..split-1]/[url=split+1..len-1]
    len = len - split - 1;
    if( len == 0 || svc->GetUrlPathType() == URL_PATH_OFF ) {
        _url_path.set_empty();
    } else {
        _url_path.append( str+(split+1), len );
    }
    name.resize( split );
    break;
      } else {
    svc->Release();
    svc = NULL;
      }
  }
    }
}
if( split == 0 ) {
    return FALSE;
}
  }
  if( svc != NULL ) {
_uid = svc->GetUid();
stmt = svc->LockStmt();
if( stmt != NULL ) {
    p_expr  stmtstr;
    uint32  len;
    stmtstr = Prep_to_str( NULL, stmt );
    stmtstr = an_ExprBuilder::GblBuilder.DB_Find_expr( stmtstr, FALSE );
    len = (uint32) stmtstr->v.str->length();
    _parms = new HttpString( len + 1 );
    {
  // Copy string to _parms
  DbStrlStream  s( *stmtstr->v.str, _CurrentConnection );
  s.get( (a_byte *) _parms->str(), (uint32) stmtstr->v.str->length() );
  _parms->resize( len );
    }
    DE_Free_expr( stmtstr );
}
svc->UnlockStmt();
result = TRUE;
  }
```

```
    _service = svc;
    return result;
}

// httppresxml
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
// ********************************************************************
// Copyright 2002-2003 iAnywhere Solutions, Inc.  All rights reserved.
// ********************************************************************
#include "httppres.h"
#include "httpstring.h"
#include "dbusrtyp.h"
#include "httputil.h"
  // #include "dblangstring.hpp"
#include "cachecarver.hpp"
#include "dbvector.h"
typedef struct _a_col_name {
    struct _a_col_name * next;
    uint16      len;
    char        name[2];      // a variable length
} a_col_name, *p_col_name;
//**********************************************************************
class HttpPresXML : public HttpPres {
protected:
    UTCollation *   _col;      // db's collation
    a_bool      _has_xml;      // db has XML typeid /
    uint16      _xml_typeid;
    a_bool      _saw_xml;      // we saw some XML columns
    a_bool      _do_xml_formatting;    // we need to do the formatting
    CacheCarver *   _carver;
    p_col_name      _first_col;
    p_col_name      _current;    // "cur" column i.e. column[_cur_idx]
    uint16          _cur_idx;
public:
    HttpPresXML( HttpOStream * ostream, UTCollation * col );
    virtual ~HttpPresXML();
protected:
    virtual void AddColumn(
      char *    table_name,
      char *    coln_name,
      uint16    coln_namelen,
      uint32    asa_usertype );
    virtual void BeginDoc( void );
    virtual void BeginResultSet( void );  // start of result set (table)
    virtual void BeginRow( void );
    virtual void BeginColumn( void );
    virtual void EndColumn( void );
    virtual void EndRow( void );
    virtual void EndResultSet( void );    // end result set (table)
    virtual void EndDoc( void );
    virtual void SendColumnValue( void * data, uint32 len );
    virtual void SendColumnMultiPiece( void * data, uint32 len );
    virtual void SendSQLError( void * errmsg, size_t len );
    virtual void NoContentDocBody( void );  // called when doc has no content
};
//**********************************************************************
HttpPres * New_HttpPresXML( HttpOStream * ostream, UTCollation * col )
/*********************************************************************/
{
    return new HttpPresXML( ostream, col );
}
HttpPresXML::HttpPresXML( HttpOStream * ostream, UTCollation * col )
    : HttpPres( ostream, col )
    , _has_xml( FALSE )
    , _xml_typeid( 0 )
    , _saw_xml( FALSE )
    , _do_xml_formatting( FALSE )
    , _carver( NULL )
    , _first_col( NULL )
    , _current( NULL )
    , _cur_idx( 0 )
/******************************/
{
    a_user_type * utype = FindUserType( "xml" );
    if( utype != NULL ) {
  _has_xml    = TRUE;
  _xml_typeid = utype->type_id;
    } else {
  //  SetPresStatus( PRES_NO_XML_USERTYPE );
  // we will format the XML output in this code
  _do_xml_formatting = TRUE;
    }
    _wants_null_values = FALSE;   // XML does not show null values
}
HttpPresXML::~HttpPresXML()
/************************/
{
    if( _carver != NULL ) {
  delete _carver;
    }
}
//**********************************************************************
// Document generation routines
//**********************************************************************
```

```
void HttpPresXML::AddColumn(
      char *    table_name,
      char *    coln_name,
      uint16    coln_namelen,
      uint32    asa_usertype )
/****************************************************/
{
    _unused( table_name );
    _unused( coln_name );
    _unused( coln_namelen );
    if( _carver == NULL ) {
  _carver    = new CacheCarver( NULL );
    }
    // check that the column name is valid XML name
    // if the user did not format the output as XML
    // then we are only going to allow "nice" ascii names
    int    i;
    int    k;
    p_col_name  p;
    char  cname[256];
    for( i=0; i<coln_namelen; i++ ) {
  if( coln_name[i] >= 'a' && coln_name[i] <= 'z' ) continue;
  if( coln_name[i] >= 'A' && coln_name[i] <= 'Z' ) continue;
  if( coln_name[i] == '_' || coln_name[i] == ':' ) continue;
  if( i == 0 ) break;
  if( coln_name[i] >= '0' && coln_name[i] <= '9' ) continue;
  if( coln_name[i] == '-' || coln_name[i] == '.' ) continue;
  break;
    }
    if( i < coln_namelen || coln_namelen == 0 ) {
  // not a valid name - make one up
  sprintf( cname, "_%d", _descriptor_count );
  coln_name    = cname;
  coln_namelen = _strlen( cname );
    }
    // make sure that the column name is unique
    for( k=0,i=0,p=_first_col; p!=NULL; ) {
  if( p->len == coln_namelen ) {
      if( _strnieq( p->name, coln_name, coln_namelen ) ) {
    // duplicate name - use a unique name
    sprintf( cname, ((i==0)?"_%d_%d":"_%d_%d_%d"), _descriptor_count, k+1, i++ );
    coln_name    = cname;
    coln_namelen = _strlen( cname );
    // start over again to make sure our new name is now unique
    k = 0;
    p = _first_col;
      }
  }
  k++;
  p = p->next;
    }
    // when we get here, the name is unique and so add it after _current
    p = (p_col_name)_carver->alloc( AL_MEMORY, sizeof(a_col_name)+coln_namelen );
    memcpy( p->name, coln_name, coln_namelen );
    p->next = NULL;
    p->len  = coln_namelen;
    if( _first_col == NULL ) {
  _first_col = p;
  _cur_idx   = 0;
    } else {
  _current->next = p;
  _cur_idx ++;
    }
    _current = p;
    if( _has_xml && asa_usertype == _xml_typeid ) {
  _saw_xml = TRUE;
    }
    if( _descriptor_count == 1 && asa_usertype == _xml_typeid ) {
  // we only expect to get one XML column ...
  // if we don't, then we will format the columns into XML ourselves
    } else {
  //  SetPresStatus( PRES_NOT_XML_RESULTSET );
  // force formatting of the output in this code
  _do_xml_formatting = TRUE;
    }
}
void HttpPresXML::BeginDoc( void )
/*******************************/
{
    PutAsc( "<?xml version=\"1.0\" ?>\n" );
}
void HttpPresXML::BeginResultSet( void )
/*************************************/
{
    PutAsc( "<root>\n" );
}
void HttpPresXML::BeginRow( void )
/*******************************/
{
    if( _do_xml_formatting ) {
  PutAsc( "<row" );
  _current = _first_col;
  _cur_idx = 0;
    }
}
void HttpPresXML::BeginColumn( void )
/**********************************/
```

```
{
    if( _do_xml_formatting ) {
  for(; _cur_idx < _col_ctr; _cur_idx++ ) {
      _assertD( _current != NULL );
      _current = _current->next;
  }
  _assertD( _current != NULL );
  PutAsc( ' ' );
  PutData( _current->name, _current->len, HF_COLNAME );
  PutAsc( "=\"" );
    }
}
void HttpPresXML::EndColumn( void )
/*********************************/
{
    if( _do_xml_formatting ) {
  PutAsc( "" );
    }
}
void HttpPresXML::EndRow( void )
/******************************/
{
    if( _do_xml_formatting ) {
  PutAsc( "/>\n" );
    }
}
void HttpPresXML::EndResultSet( void )
/************************************/
{
    PutAsc( "</root>\n" );
}
void HttpPresXML::EndDoc( void )
/******************************/
{
}
void HttpPresXML::SendColumnValue( void * data, uint32 len )
/**********************************************************/
{
    if( _do_xml_formatting ) {
  if( len > 0 ) {
      PutData( (char *)data, (size_t)len, HF_ENC );  // db data is cs-conv up in engine
  }
    } else {
  // no encoding or conversion is required because the engine already did it
  PutData( (char *)data, (size_t)len, HF_NONE );    // data is already XML
    }
}
```

```
void HttpPresXML::SendColumnMultiPiece( void * data, uint32 len )
/***************************************************************/
{
    if( _do_xml_formatting ) {
	if( len > 0 ) {
	    PutData( (char *)data, (size_t)len, HF_ENC );  // db data is cs-conv up in engine
	}
    } else {
	// no encoding or conversion is required because the engine already did it
	PutData( (char *)data, (size_t)len, HF_NONE );    // data is already XML
    }
}
void HttpPresXML::SendSQLError( void * errmsg, size_t len )
/*********************************************************/
{
    PutAsc( "<SQLerror message=\"" );
    PutData( (char *)errmsg, len, HF_ENC|HF_CONV );
    PutAsc( "\"/> " );
}
void HttpPresXML::NoContentDocBody( void )
/****************************************/
{
    // in this case, we want to just dump the headers without any rows
    BeginResultSet();
    EndResultSet();
}


// sa_set_http_header
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
static a_ptrint
sa_set_http_header( a_queue *parm_q )
/***********************************/
{
    char *  fldname;
    char *  val;
    GetParms( parm_q, &fldname, &val );
#if defined( HTTP_SUPPORT )
    if( !_CurrentConnection->SetHTTPHeaderField( fldname, val ) ) {
	dbi_sql_errors( SQLSTATE_INVALID_HTTP_HEADER_SETTING, fldname );
	return( -1 );
    }
#endif
    return( 0 );
}


```

```
// sa_set_http_option
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
static a_ptrint
sa_set_http_option( a_queue *parm_q )
/***********************************/
{
    char *  optname;
    char *  val;
    GetParms( parm_q, &optname, &val );
#if defined( HTTP_SUPPORT )
    if( !_CurrentConnection->SetHTTPOption( optname, val ) ) {
  dbi_sql_errors( SQLSTATE_INVALID_HTTP_OPTION_SETTING, optname );
  return( -1 );
    }
#endif
    return( 0 );
}


// sethttpheaderfield
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
a_bool Connection::SetHTTPHeaderField( char * fldname, char * val )
/****************************************************************/
{
    a_bool  result = FALSE;
    if( http_conn != NULL && fldname != NULL && *fldname != '\0' ) {
  result = http_conn->GetProtocol()->SetResponseHeader( fldname, val
== NULL ? "" : val );
    }
    return( result );
}


// sethttpoption
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
a_bool Connection::SetHTTPOption( char * optname, char * val )
/********************************************************/
{
    a_bool  result = FALSE;
    if( http_conn != NULL && optname != NULL && *optname != '\0' ) {
  result = http_conn->GetProtocol()->SetHTTPOption( optname,
    (char *)( val == NULL ? "" : val ) );
    }
    return( result );
}
a_bool HttpProtocol::SetHTTPOption( char * optname, char * value )
/**************************************************************/
{
    // Validate option names [and values].
    if( _strieq( optname, "CharsetConversion" ) ) {
```

```
  if( _strieq( value, "ON" ) ) {
     _connection->GetStream()->set_translation_wanted( TRUE );
  } else if( _strieq( value, "OFF" ) ) {
     _connection->GetStream()->set_translation_wanted( FALSE );
  } else {
      return( FALSE );
  }
    } else {
  return( FALSE );
    }
    _options.Set( optname, value );
    return( TRUE );
}


// setresponseheader
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
a_bool HttpProtocol::SetResponseHeader( const char * key, const char * value )
/***************************************************************************/
{
    _assertD( key != NULL );
    if( *key == '@' ) {
  // special values
  if( _strieq( key, "@HttpStatus" ) ) {
      // value had better be the numeric status code
      int v = atoi( value );
      int i = StatusLineIndex( v );
      if( i == LEVEL_600 ) {
    // invalid value
    return FALSE;
      }
      SetHttpStatus( (HttpStatus)v );
      return TRUE;
  }
    }
    // check that the key and value consists of valid HTTP characters
    const char * s;
#define IS_HTTP_TOKEN_CHAR( c ) ( (c)>' ' && (c)<='~' && !IsHttpSeparator( c ) )
#define VALID_HTTP_KEY_CHAR( c ) IS_HTTP_TOKEN_CHAR( c )
#define VALID_HTTP_VAL_CHAR( c ) ( ((c)>=' ' && (c)<='~') || (c)=='\t' )
    // TBD: we need to handle LWS (continuation lines) in the field values
    for( s=key; *s != '\0'; s++ ) {
  if( !VALID_HTTP_KEY_CHAR( *s ) ) {
      return FALSE;
  }
```

```
    }
    for( s=value; *s != '\0'; s++ ) {
  if( !VALID_HTTP_VAL_CHAR( *s ) ) {
      return FALSE;
  }
    }
    _response.Set( key, value );
    return TRUE;
}


```